Package ‘gnn’

March 4, 2020

Version 0.0-2

Title Generative Neural Networks

Description Tools to set up, train, store, load, investigate and analyze
generative neural networks. In particular, functionality for
generative moment matching networks is provided.
See Hofert, Prasad, Zhu (2018) <arXiv:1811.00683>
for more details.

Maintainer Marius Hofert <`marius.hofert@uwaterloo.ca`>

Depends R (>= 3.5.0)

Imports keras, tensorflow, qrng, methods, tools

Suggests copula, qrmtools, qrmdata, latticeExtra, RnavGraphImageData

Enhances

License GPL (>= 3)

SystemRequirements TensorFlow (https://www.tensorflow.org/)

NeedsCompilation no

Repository CRAN

Encoding UTF-8

Date/Publication 2020-03-03 23:40:02 UTC

Author Marius Hofert [aut, cre] (<https://orcid.org/0000-0001-8009-4665>),
Avinash Prasad [aut]

R topics documented:

`GMMN_model` *Generative Moment Matching Network*

Description

Setup of a generative moment matching network (GMMN) model.

Usage

```
GMMN_model(dim, activation = c(rep("relu", length(dim) - 2), "sigmoid"),
           batch.norm = FALSE, dropout.rate = 0, nGPU = 0, ...)
```

Arguments

`dim`	`numeric` vector of length at least two, giving the dimensions of the input layer, the hidden layer(s) (if any) and the output layer (in this order).
`activation`	`character` vector of length `length(dim) -1` specifying the activation functions for all hidden layers and the output layer (in this order); note that the input layer does not have an activation function.
`batch.norm`	`logical` indicating whether batch normalization layers are to be added after each hidden layer.
`dropout.rate`	`numeric` value in [0,1] specifying the fraction of input to be dropped; see the rate parameter of `layer_dropout()`. Note that only if positive, dropout layers are added after each hidden layer.
`nGPU`	non-negative `integer` specifying the number of GPUs available if the GPU version of TensorFlow is installed. If positive, a (special) multiple GPU model for data parallelism is instantiated. Note that for multi-layer perceptrons on a few GPUs, this model does not yet yield any scale-up computational factor (in fact, currently very slightly negative scale-ups are likely due to overhead costs).
`...`	additional arguments passed to `loss()`.

Value

`GMMN_model()` returns a list with components

`model`**:** GMMN model (a **keras** object inheriting from the classes `"keras.engine.training.Model"`, `"keras.engine.network.Network"`, `"keras.engine.base_layer.Layer"` and `"python.builtin.object"`).

`type`**:** `character` string indicating the type of model (`"GMMN"`).

`dim`**:** see above.

`activation`**:** see above.

`batch.norm`**:** see above.

`dropout.rate`**:** see above.

`dim.train`**:** dimension of the training data (`NA` unless trained).

`batch.size`**:** batch size (`NA` unless trained).

`nepoch`**:** number of epochs (`NA` unless trained).

Author(s)

Marius Hofert and Avinash Prasad

References

Li, Y., Swersky, K. and Zemel, R. (2015). Generative moment matching networks. *Proceedings of Machine Learning Research*, **37** (International Conference on Maching Learning), 1718–1727. See http://proceedings.mlr.press/v37/li15.pdf (2019-08-24)

Dziugaite, G. K., Roy, D. M. and Ghahramani, Z. (2015). Training generative neural networks via maximum mean discrepancy optimization. *AUAI Press*, 258–267. See http://www.auai.org/uai2015/proceedings/papers/230.p
(2019-08-24)

See Also

`VAE_model()`

Examples

```
 # to avoid win-builder error "Error: Installation of TensorFlow not found"
## Example model with a 5d input, 300d hidden and 4d output layer
str(GMMN_model(c(5, 300, 4)))
```

`GMMN_trained` *Trained Generative Moment Matching Networks*

Description

Trained generative moment matching networks (GMMNs); see also the demo `GMMN_QMC` or the vignette `GMMN_QMC`.

Usage

```
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.25")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.75")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.25")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.75")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_C_tau_0.5_rot90_t4_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_G_tau_0.5_rot90_t4_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_MO_0.75_0.6_rot90_t4_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_MO_0.75_0.6")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.25")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5")
data("GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.75")
data("GMMN_dim_3_300_3_ntrn_60000_nbat_5000_nepo_300_NC21_tau_0.25_0.5")
data("GMMN_dim_3_300_3_ntrn_60000_nbat_5000_nepo_300_NG21_tau_0.25_0.5")
data("GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5")
data("GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5")
data("GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_NC23_tau_0.25_0.5_0.75")
data("GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_NG23_tau_0.25_0.5_0.75")
data("GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5")
data("GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5")
data("GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5")
data("GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_NC55_tau_0.25_0.5_0.75")
data("GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_NG55_tau_0.25_0.5_0.75")
data("GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5")
```

Format

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.25 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Clayton copula (with parameter chosen such that Kendall's tau equals 0.25).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Clayton copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_C_tau_0.75 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Clayton copula (with parameter chosen such that Kendall's tau equals 0.75).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.25 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Gumbel copula (with parameter chosen such that Kendall's tau equals 0.25).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Gumbel copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_G_tau_0.75 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate Gumbel copula (with parameter chosen such that Kendall's tau equals 0.75).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_C_tau_0.5_rot90_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate half-half mixture of a Clayton copula (with parameter chosen such that Kendall's tau equals 0.5) and a rotated (by 90 degree) t copula (with 4 degrees of freedom and correlation parameter chosen such that Kendall's tau equals 0.5); see `vignette("GMMN_QRNG",package = "gnn")`.

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_G_tau_0.5_rot90_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate half-half mixture of a Gumbel copula (with parameter chosen such that Kendall's tau equals 0.5) and a rotated (by 90 degree) t copula (with 4 degrees of freedom and correlation parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_MO_0.75_0.6_rot90_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a bivariate half-half mixture of a Marshall–Olkin copula (with $\alpha_1 = 0.75$ and $\alpha_2 = 0.60$) and a rotated (by 90 degree) t copula (with 4 degrees of freedom and correlation parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_MO_0.75_0.6 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a Marshall–Olkin copula (with $\alpha_1 = 0.75$ and $\alpha_2 = 0.60$).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.25 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a two-dimensional t copula (with 4 degrees of freedom and equi-correlation parameter chosen such that Kendall's tau equals 0.25).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a two-dimensional t copula (with 4 degrees of freedom and equi-correlation parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.75 `raw` R object representing a GMMN (input and output layer are two-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a two-dimensional t copula (with 4 degrees of freedom and equi-correlation parameter chosen such that Kendall's tau equals 0.75).

GMMN_dim_3_300_3_ntrn_60000_nbat_5000_nepo_300_NC21_tau_0.25_0.5 `raw` R object representing a GMMN (input and output layer are three-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a three-dimensional nested Clayton copula (with sector dimensions 2 and 1, corresponding Kendall's tau 0.5 within the first sector and Kendall's tau 0.25 between the two sectors).

GMMN_dim_3_300_3_ntrn_60000_nbat_5000_nepo_300_NG21_tau_0.25_0.5 `raw` R object representing a GMMN (input and output layer are three-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a three-dimensional nested Gumbel copula (with sector dimensions 2 and 1, corresponding Kendall's tau 0.5 within the first sector and Kendall's tau 0.25 between the two sectors).

GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5 `raw` R object representing a GMMN (input and output layer are five-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a five-dimensional Clayton copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5 `raw` R object representing a GMMN (input and output layer are five-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a five-dimensional Gumbel copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_NC23_tau_0.25_0.5_0.75 `raw` R object representing a GMMN (input and output layer are five-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a five-dimensional nested Clayton copula (with sector dimensions 2 and 3, corresponding Kendall's tau 0.5 and 0.75, and Kendall's tau 0.25 between the two sectors).

GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_NG23_tau_0.25_0.5_0.75 `raw` R object representing a GMMN (input and output layer are five-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a five-dimensional nested Gumbel copula (with sector dimensions 2 and 3, corresponding Kendall's tau 0.5 and 0.75, and Kendall's tau 0.25 between the two sectors); see `vignette("GMMN_QRNG",package = "gnn")`.

GMMN_dim_5_300_5_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are five-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a five-dimensional t copula (with 4 degrees of freedom and equi-correlation parameter chosen such that Kendall's tau equals 0.5); see `vignette("GMMN_QRNG",package = "gnn")`.

GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_C_tau_0.5 `raw` R object representing a GMMN (input and output layer are 10-dimensional, the single hiddenlayer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a 10-dimensional Clayton copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_G_tau_0.5 `raw` R object representing a GMMN (input and output layer are 10-dimensional, the single hiddenlayer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a 10-dimensional Gumbel copula (with parameter chosen such that Kendall's tau equals 0.5).

GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_NC55_tau_0.25_0.5_0.75 `raw` R object representing a GMMN (input and output layer are 10-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a 10-dimensional nested Clayton copula (with sector dimensions 5 and 5, corresponding Kendall's tau 0.5 and 0.75, and Kendall's tau 0.25 between the two sectors).

GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_NG55_tau_0.25_0.5_0.75 `raw` R object representing a GMMN (input and output layer are 10-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a 10-dimensional nested Gumbel copula (with sector dimensions 5 and 5, corresponding Kendall's tau 0.5 and 0.75, and Kendall's tau 0.25 between the two sectors).

GMMN_dim_10_300_10_ntrn_60000_nbat_5000_nepo_300_t4_tau_0.5 `raw` R object representing a GMMN (input and output layer are 10-dimensional, the single hidden layer is 300-dimensional) trained on 60000 pseudo-samples (with batch size 5000 and 300 epochs) from a

10-dimensional $t$ copula (with 4 degrees of freedom and equi-correlation parameter chosen such that Kendall's tau equals 0.5).

Author(s)

Marius Hofert and Avinash Prasad

Source

GPU server with NVIDIA Tesla P100 GPUs.

References

Hofert, M., Prasad, A. and Zhu, M. (2018). Quasi-Monte Carlo for multivariate distributions via generative neural networks. (See https://arxiv.org/abs/1811.00683 for an early version)

See Also

`GMMN_model()`, `to_callable()`

Examples

```
 # to avoid win-builder error "Error: Installation of TensorFlow not found"
## Load a trained GMMN (see train_once())
NNname <- "GMMN_dim_2_300_2_ntrn_60000_nbat_5000_nepo_300_eqmix_C_tau_0.5_rot90_t4_tau_0.5"
NN <- read_rda(NNname, package = "gnn")
GMMN1 <- to_callable(NN)
str(GMMN1)

## Alternative
NNnm <- data(list = NNname)
GMMN2 <- to_callable(get(NNnm))
str(GMMN2)

## Check (the check-able components)
stopifnot(identical(GMMN1[names(GMMN1) != "model"],
                    GMMN2[names(GMMN2) != "model"]))

## Evaluate
set.seed(271)
N.prior <- matrix(rnorm(2000 * 2), ncol = 2)
X <- predict(GMMN1[["model"]], x = N.prior)
plot(X, xlab = expression(X[1]), ylab = expression(X[2]))
```

human_time *Time Measurement with Human-Readable Units*

Description

system.time() with human-readable output.

Usage

```
human_time(..., digits = 2)
```

Arguments

...	arguments passed to the underlying system.time().
digits	for rounding the output; see round().

Value

Timings with units indicated.

Author(s)

Marius Hofert

Examples

```
human_time(Sys.sleep(1))
```

loss *Loss Function*

Description

Implementation of various loss functions to measure statistical discrepancy between two datasets.

Usage

```
loss(x, y, type = c("MSE", "binary.cross", "MMD"), ...)
```

Arguments

x	2d tensor with shape (batch size, dimension of input dataset).
y	2d tensor with shape (batch size, dimension of input dataset).
type	character string indicating the type of loss used. Currently available are the mean squared error ("MSE"), binary cross entropy ("binary.cross") and (kernel) maximum mean discrepancy ("MMD").
...	additional arguments passed to the underlying loss function; at the moment, this is only affects type = "MMD" for which "bandwidth" can be provided.

Value

`loss()` returns a 0d tensor containing the loss.

Author(s)

Marius Hofert and Avinash Prasad

References

Kingma, D. P. and Welling, M. (2014). Stochastic gradient VB and the variational auto-encoder. *Second International Conference on Learning Representations (ICLR)*. See https://keras.rstudio.com/articles/examples/variat

See Also

`GMMN_model()` and `VAE_model()` where `loss()` is used.

`rda` *Check Existence, Read, Save and Rename .rda Files and their Objects*

Description

Check Existence, Read, Save and Rename `.rda` Files and their Objects.

Usage

```
exists_rda(file, names, package = NULL)
read_rda(file, names, package = NULL)
save_rda(..., file, names = NULL)
rename_rda(oldname, oldfile = paste0(oldname, collapse = "_"),
           newname, newfile = paste0(newname, collapse = "_", ".rda"),
   package = NULL)
```

Arguments

`file`

exists_rda() `character` string (with or without ending `.rda`) specifying the name of the file to check existence of (if `pacakge = NULL`) or in (otherwise).

read_rda() `character` string (with or without ending `.rda`) specifying the file to read from.

save_rda() `character` string (with or without ending `.rda`) specifying the file to save to.

`names`

exists_rda() `character` vector of names of objects to be checked for existence.

read_rda() `character` vector of names of objects to be read. If not provided, a name is constructed from `file`.

save_rda() `character` vector of names under which the objects in `...` are saved in `file`. If `NULL`, the names of the objects provided by ... are taken as default values.

`package`	**exists_rda()** package name in which to check or `NULL` (the default) in which case the current working directory is checked. **read_rda(), rename_rda()** package name from which to load the objects or `NULL` (the default) in which case the current working directory is searched.
`...`	any number of R objects.
`oldname`	`character` string specifying the object to be read.
`oldfile`	file name (with or without ending `.rda`) specifying from which the object named `oldname` is read.
`newname`	`character` string specifying the new name under which the object is to be saved.
`newfile`	file name (with ending `.rda`) specifying where the object named `oldname` is saved under the name `newname`.

Value

exists_rda() `logical` indicating whether the `.rda` file `file` exists (if `names` is not provided) or whether the objects with names `names` exist inside `file` (if `names` is provided).

read_rda() the object read from the `.rda`.

save_rda() nothing (generated an `.rda` by side-effect).

rename_rda() nothing (generated an `.rda` by side-effect).

Author(s)

Marius Hofert

See Also

See the underlying functions `load()`, `data()` and `save()` (among others).

`rm_ext` *Remove a File Extension*

Description

Fixes the removal of file extensions of `file_path_sans_ext()` in the case where file names contain digits after the last dot (which is often used to incorporate numeric numbers into file names).

Usage

```
rm_ext(x)
```

Arguments

`x`	file name(s) with extension(s) to be stripped off.

Value

The file name without its extension (if the file name had an extension).

Author(s)

Marius Hofert

Examples

```
myfilepath1 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75.rda"
myfilepath2 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75"
myfilepath3 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75."
myfilepath4 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75._"
myfilepath5 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75._*.rda"
library(tools)
file_path_sans_ext(myfilepath2) # fails (only case)

stopifnot(rm_ext(myfilepath1) == file_path_sans_ext(myfilepath1))
stopifnot(rm_ext(myfilepath2) == myfilepath2)
stopifnot(rm_ext(myfilepath3) == file_path_sans_ext(myfilepath3))
stopifnot(rm_ext(myfilepath4) == file_path_sans_ext(myfilepath4))
stopifnot(rm_ext(myfilepath5) == file_path_sans_ext(myfilepath5))
```

`to_savable_callable` *Convert GNN objects to Savable or Callable Ones*

Description

Keras objects cannot be saved like other R objects. The auxiliary functions `to_savable()` and `to_callable()` address this issue.

Usage

```
to_savable(gnn)
to_callable(gnn)
```

Arguments

`gnn`	GNN object.

Details

For GMMNs, `to_savable()` calls `serialize_model()` and `to_callable()` calls `unserialize_model()`.

For VAEs, `to_savable()` is (indirectly) based on `save_model_weights_hdf5()` and `to_callable()` on `load_model_weights_hdf5()`; one cannot work with `serialize_model()` or `unserialize_model()` in this case because of the involved `layer_lambda()`.

See the source code for more details.

Value

`to_savable()`: The GNN object with **keras** components replaced by savable ones.

`to_callable()`: The GNN object with certain components replaced by **keras** objects.

Author(s)

Marius Hofert

`trafos_componentwise` *Data Transformations for Training or Sampling*

Description

Transformations applied to each marginal component sample to map given data to a different range.

Usage

```
range_trafo(x, lower, upper, inverse = FALSE)
logis_trafo(x, mean = 0, sd = 1, slope = 1, intercept = 0, inverse = FALSE)
```

Arguments

`x`	(n, d)-matrix of data (typically before training or after sampling).
`lower`	value or d-vector typically containing the smallest value of each column of `x`.
`upper`	value or d-vector typically containing the largest value of each column of `x`.
`mean`	value or d-vector.
`sd`	value or d-vector.
`slope`	value or d-vector of slopes of the linear transformations applied after applying `plogis()` (before applying `qlogis()` if `inverse = TRUE`).
`intercept`	value or d-vector of intercepts of the linear transformations applied after applying `plogis()` (before applying `qlogis()` if `inverse = TRUE`).
`inverse`	`logical` indicating whether the inverses of the respective transformations are to be computed (typically used after generating data from a neural network trained on data transformed with the respective transformation and `inverse = FALSE`).

Value

An object as `x` containing the componentwise transformed data.

Author(s)

Marius Hofert

Examples

```
## Generate data
n <- 100
set.seed(271)
x <- cbind(rnorm(n), (1-runif(n))^(-1/2)-1) # normal and Pareto(2) margins
plot(x)

## Range transformation
ran <- apply(x, 2, range) # column j = range of the jth column of x
x.ran <- range_trafo(x, lower = ran[1,], upper = ran[2,]) # marginally transform to [0,1]
plot(x.ran) # => now range [0,1] but points a bit clustered around small y-values
x. <- range_trafo(x.ran, lower = ran[1,], upper = ran[2,], inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation
x.logis <- logis_trafo(x) # marginally transform to [0,1] via plogis()
plot(x.logis) # => y-range is [1/2, 1] which can be harder to train
x. <- logis_trafo(x.logis, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation with scaling to all of [0,1] in the second coordinate
x.logis.scale <- logis_trafo(x, slope = 2, intercept = -1)
plot(x.logis.scale) # => now y-range is scaled to [0,1]
x. <- logis_trafo(x.logis.scale, slope = 2, intercept = -1, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation with sample mean and standard deviation and then
## transforming the range to [0,1] with a range transformation (note that
## slope = 2, intercept = -1 would not help here as the y-range is not [1/2, 1])
mu <- colMeans(x)
sig <- apply(x, 2, sd)
x.logis.fit <- logis_trafo(x, mean = mu, sd = sig) # marginally plogis(, location, scale)
plot(x.logis.fit) # => y-range is not [1/2, 1] => use range transformation
ran <- apply(x.logis.fit, 2, range)
x.logis.fit.ran <- range_trafo(x.logis.fit, lower = ran[1,], upper = ran[2,])
plot(x.logis.fit.ran) # => now y-range is [1/2, 1]
x. <- logis_trafo(range_trafo(x.logis.fit.ran, lower = ran[1,], upper = ran[2,],
                              inverse = TRUE),
                  mean = mu, sd = sig, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Note that for heavy-tailed data, plogis() can fail to stay inside (0,1)
## even with adapting to sample mean and standard deviation. We now present
## a case where we see that using a fitted logistic distribution function
## is *just* good enough to numerically keep the data inside (0,1).
set.seed(271)
x <- cbind(rnorm(n), (1-runif(n))^(-2)-1) # normal and Pareto(1/2) margins
plot(x) # => heavy-tailed in y-coordinate
## Transforming with standard logistic distribution function
x.logis <- logis_trafo(x)
stopifnot(any(x.logis[,2] == 1))
## => There is value numerically indistinguishable from 1 to which applying
##    the inverse transform will lead to Inf
stopifnot(any(is.infinite(logis_trafo(x.logis, inverse = TRUE))))
## Now adapt the logistic distribution to share the mean and standard deviation
## with the data
mu <- colMeans(x)
sig <- apply(x, 2, sd)
x.logis.scale <- logis_trafo(x, mean = mu, sd = sig)
stopifnot(all(x.logis.scale[,2] != 1)) # => no values equal to 1 anymore

## Alternatively, log() the data first, thus working with a log-logistic
## distribution as transformation
lx <- cbind(x[,1], log(x[,2])) # 2nd coordinate only
lmu <- c(mu[1], mean(lx[,2]))
lsig <- c(sig[1], sd(lx[,2]))
x.llogis <- logis_trafo(lx, mean = lmu, sd = lsig)
x. <- logis_trafo(x.llogis, mean = lmu, sd = lsig, inverse = TRUE)
x.. <- cbind(x.[,1], exp(x.[,2])) # undo log()
stopifnot(all.equal(x.., x))
```

`trafos_dimreduction` *Dimension-Reduction Transformations for Training or Sampling*

Description

Dimension-reduction transformations applied to an input data matrix. Currently on the principal component transformation and its inverse.

Usage

```
PCA_trafo(x, mu, Gamma, inverse = FALSE, ...)
```

Arguments

`x`	(n, d)-matrix of data (typically before training or after sampling). If `inverse = FALSE`, then, conceptually, an (n, d)-matrix with $1 \le k \le d$, where d is the dimension of the original data whose dimension was reduced to k.
`mu`	if `inverse = TRUE`, a d-vector of centers, where d is the dimension to transform `x` to.
`Gamma`	if `inverse = TRUE`, a (d, k)-matrix with k at least as large as `ncol(x)` containing the k orthonormal eigenvectors of a covariance matrix sorted in decreasing order of their eigenvalues; in other words, the columns of `Gamma` contain principal axes or loadings. If a matrix with k greater than `ncol(x)` is provided, only the first k-many are considered.
`inverse`	`logical` indicating whether the inverse transformation of the principal component transformation is applied.
`...`	additional arguments passed to the underlying `prcomp()`.

Details

Conceptually, the principal component transformation transforms a vector $\boldsymbol{X}$ to a vector $\boldsymbol{Y}$ where $\boldsymbol{Y} = \Gamma^T(\boldsymbol{X} - \boldsymbol{\mu})$, where $\boldsymbol{\mu}$ is the mean vector of $\boldsymbol{X}$ and Γ is the (d, d)-matrix whose columns contains the orthonormal eigenvectors of `cov(X)`.

The corresponding (conceptual) inverse transformation is $\boldsymbol{X} = \boldsymbol{\mu} + \Gamma\boldsymbol{Y}$.

See McNeil et al. (2015, Section 6.4.5).

Value

If `inverse = TRUE`, the transformed data whose rows contain $\boldsymbol{X} = \boldsymbol{\mu} + \Gamma\boldsymbol{Y}$, where Y is one row of `x`. See the details below for the notation.

If `inverse = FALSE`, a `list` containing:

`PCs`: (n, d)-matrix of principal components.

`cumvar`: cumulative variances; the jth entry provides the fraction of the explained variance of the first j principal components.

`sd`: sample standard deviations of the transformed data.

`lambda`: eigenvalues of `cov(x)`.

`mu`: d-vector of centers of `x` (see also above) typically provided to `PCA_trafo(,inverse = TRUE)`.

`Gamma`: (d, d)-matrix of principal axes (see also above) typically provided to `PCA_trafo(,inverse = TRUE)`.

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
## Generate data
library(copula)
set.seed(271)
X <- qt(rCopula(1000, gumbelCopula(2, dim = 10)), df = 3.5)
pairs(X, gap = 0, pch = ".")

## Principal component transformation
PCA <- PCA_trafo(X)
Y <- PCA$PCs
PCA$cumvar[3] # fraction of variance explained by the first 3 principal components
which.max(PCA$cumvar > 0.9) # number of principal components it takes to explain 90%

## Biplot (plot of the first two principal components = data transformed with
## the first two principal axes)
plot(Y[,1:2])
```

```
## Transform back and compare
X. <- PCA_trafo(Y, mu = PCA$mu, Gamma = PCA$Gamma, inverse = TRUE)
stopifnot(all.equal(X., X))

## Note: One typically transforms back with only some of the principal axes
X. <- PCA_trafo(Y[,1:3], mu = PCA$mu, # mu determines the dimension to transform to
                Gamma = PCA$Gamma, # must be of dim. (length(mu), k) for k >= ncol(x)
                inverse = TRUE)
stopifnot(dim(X.) == c(1000, 10))
## Note: We (typically) transform back to the original dimension.
pairs(X., gap = 0, pch = ".") # pairs of back-transformed first three PCs
```

training *Functions for Training of Generative Neural Networks*

Description

Functions for training generative neural networks.

Usage

```
train(gnn, data, batch.size, nepoch, verbose = 3, ...)
train_once(gnn, data, batch.size, nepoch,
           file, name = rm_ext(basename(file)), package = NULL, ...)
```

Arguments

`gnn`	GNN object as created by `GMMN_model()` or `VAE_model()`.
`data`	(n, d)-matrix containing the n d-dimensional observations of the training data.
`batch.size`	number of samples used per stochastic gradient step.
`nepoch`	number of epochs (one epoch equals one pass through the complete training dataset while updating the GNN's parameters through stochastic gradient steps).
`verbose`	see `fit.keras.engine.training.Model()`.
`file`	`character` string (with or without ending `.rda`) specifying the file to save the trained GNN to.
`name`	name under which the trained GNN is saved in `file`.
`package`	name of the package from which to read the trained GNN; if `NULL` (the default) the current working directory is used.
`...`	additional arguments passed to the underlying `train()` for `train_once()` and `fit()` (which is `keras:::fit.keras.engine.training.Model()`) for `train()`.

Value

`train()`**:** The trained GNN object.

`train_once()`**:** If object name exists in `file`, `train_once()` reads it, converts it to a callable GNN object via `to_callable()` and returns it. Otherwise, `train_once()` calls `train()` to train the GNN, converts it to a savable GNN object via `to_savable()`, saves it and returns the trained GNN.

Author(s)

Marius Hofert

See Also

`GMMN_model()`, `VAE_model()`, `to_savable()`, `to_callable()`.

Examples

```
## Training data
d <- 2
P <- matrix(0.9, nrow = d, ncol = d)
diag(P) <- 1
A <- t(chol(P))
set.seed(271)
ntrn <- 60000
Z <- matrix(rnorm(ntrn * d), ncol = d)
X <- t(A %*% t(Z)) # d-dimensional equicorrelated normal
U <- apply(abs(X), 2, rank) / (ntrn + 1) # pseudo-observations of |X|
plot(U[1:2000,], xlab = expression(U[1]), ylab = expression(U[2]))

## Define the model and 'train' it
dim <- c(d, 300, d) # dimensions of the input, hidden and output layers
GMMN.mod <- GMMN_model(dim)
GMMN.trained <- train(GMMN.mod, data = U, batch.size = 500, nepoch = 2)
## Note: Obviously, in a real-world application, batch.size and nepoch
##       should be (much) larger (e.g., batch.size = 5000, nepoch = 300).

## Evaluate (roughly picks up the shape even with our bad choices of
## batch.size and nepoch)
set.seed(271)
N.prior <- matrix(rnorm(2000 * d), ncol = d)
V <- predict(GMMN.trained[["model"]], x = N.prior)
plot(V, xlab = expression(V[1]), ylab = expression(V[2]))

## Convert the trained neural network to one that can be saved
## and save it (here: to some temporary file)
GMMN.savable <- to_savable(GMMN.trained)
file <- tempfile("trained_GMMN", fileext = ".rda")
save_rda(GMMN.savable, file = file, names = "GMMN")
```

`VAE_model` *Variational Autoencoder*

Description

Setup of a variational autoencoder (VAE) model.

Usage

```
VAE_model(dim, activation = c(rep("relu", length(dim) - 2), "sigmoid"),
          batch.norm = FALSE, dropout.rate = 0,
          sd = 1, loss.type = c("MSE", "binary.cross", "MMD"), nGPU = 0, ...)
```

Arguments

`dim`	`numeric` vector of length at least two, giving the dimensions of the input layer (equal to the dimension of the output layer), the hidden layer(s) (if any) and the latent layer (in this order).
`activation`	`character` vector of length `length(dim) -1` specifying the activation functions for all hidden layers and the output layer (in this order); note that the input layer does not have an activation function.
`batch.norm`	`logical` indicating whether batch normalization layers are to be added after each hidden layer.
`dropout.rate`	`numeric` value in [0,1] specifying the fraction of input to be dropped; see the rate parameter of `layer_dropout()`. Note that only if positive, dropout layers are added after each hidden layer.
`sd`	positive `numeric` value giving the standard deviation of the normal distribution used as prior.
`loss.type`	`character` string indicating the type of reconstruction loss. Currently available are the mean squared error (`"MSE"`), binary cross entropy (`"binary.cross"`) and (kernel) maximum mean discrepancy (`"MMD"`).
`nGPU`	non-negative `integer` specifying the number of GPUs available if the GPU version of TensorFlow is installed. If positive, a (special) multiple GPU model for data parallelism is instantiated. Note that for multi-layer perceptrons on a few GPUs, this model does not yet yield any scale-up computational factor (in fact, currently very slightly negative scale-ups are likely due to overhead costs).
`...`	additional arguments passed to `loss()`.

Value

`VAE_model()` returns a list with components

`model`**:** VAE model (a **keras** object inheriting from the classes `"keras.engine.training.Model"`, `"keras.engine.network.Network"`, `"keras.engine.base_layer.Layer"` and `"python.builtin.object"`).

`encoder`**:** the encoder (a **keras** object as `model`).

`generator`**:** the generator (a **keras** object as `model`).

`type`**:** `character` string indicating the type of model (`"VAE"`).

`dim`**:** see above.

`activation`**:** see above.

`batch.norm`**:** see above.

`dropout.rate`**:** see above.

`sd`**:** see above.

`loss.type`**:** see above.

`dim.train`**:** dimension of the training data (`NA` unless trained).

`batch.size`**:** batch size (`NA` unless trained).

`nepoch`**:** number of epochs (`NA` unless trained).

Author(s)

Marius Hofert and Avinash Prasad

References

Kingma, D. P. and Welling, M. (2014). Stochastic gradient VB and the variational auto-encoder. *Second International Conference on Learning Representations (ICLR)*. See https://keras.rstudio.com/articles/examples/variat

See Also

`GMMN_model()`

Examples

```
 # to avoid win-builder error "Error: Installation of TensorFlow not found"
## Example model with a 5d input, 300d hidden and 4d output layer
str(VAE_model(c(5, 300, 4)))
```

Index